Insights Into Malware Detection via Behavioral Frequency Analysis Using Machine Learning

Aaron Walker
Department of Computer Science and Engineering
University of Nevada, Reno
Reno, U.S.A.
awalker@unr.edu

Shamik Sengupta
Department of Computer Science and Engineering
University of Nevada, Reno
Reno, U.S.A.
ssengupta@unr.edu

***Abstract*—The most common defenses against malware threats involves the use of signatures derived from instances of known malware. However, the constant evolution of the malware threat landscape necessitates defense against unknown malware, making a signature catalog of known threats insufficient to prevent zero-day vulnerabilities from being exploited. Recent research has applied machine learning approaches to identify malware through artifacts of malicious activity as observed through dynamic behavioral analysis. We have seen that these approaches mimic common malware defenses by simply offering a method of detecting known malware. We contribute a new method of identifying software as malicious or benign through analysis of the frequency of Windows API system function calls. We show that this is a powerful technique for malware detection because it generates learning models which understand the difference between malicious and benign software, rather than producing a malware signature classifier. We contribute a method of systematically comparing machine learning models against different datasets to determine their efficacy in accurately distinguishing the difference between malicious and benign software.**

***Index Terms*—Malware Detection, Malware Behavioral Analysis, Machine Learning, Dynamic Analysis, Zero-Day**

I. Introduction

Malware, or malicious software, threaten computer systems and applications at home as well as the office, infecting a host system whether that system is mobile or stationary. If the software matches a signature of a known malware, that software can be safely quarantined or deleted from the system - hopefully before any damage is done. The practice of signature-based malware detection is sufficient for protecting a system against known malware. Unfortunately, signature-based defense only works against known malware. Antivirus companies analyze newly discovered malware and release signatures to their customers to protect them against emergent threats, but until a signature is released for a newly identified malware there will be no defense against infection. Advanced Persistent Threat actors (APT) have frequently exploited long-standing security vulnerabilities for a number of years without detection. Clearly, we can see that defense involving only known threats is insufficient to ensure the confidentiality, availability, and integrity of computing systems, applications, and data. However, we believe that there is not a "one size fits all" approach to analyzing malware with machine learning.

In our research we found a lack of analysis regarding the meaning of why several different machine learning approaches can identify malware with high accuracy on specific datasets. It is not always clear how machine learning models for detecting malware address the issues of false positive and false negative classification. We show that malware and benign software contain many of the same Windows API function calls which define their behavior on a host computer. Analysis of only the presence of these function calls is therefore not sufficient to divide malware from benign software. In this paper we will show that analysis of the frequencies of these API calls provides a powerful means of understanding the difference between malicious and benign activity. Furthermore, our interest is in observing what machine learning algorithms can teach us about malware as well as our understanding of how to identify malware. Our research leads us to discover the relationship between the system calls used as features in the models. This paper presents a first step on the path to revise the way we think about the behavior of malware and what that could mean for the next generation of malware analysis systems.

Contribution: Our contributions in this paper include:

- We show that the analysis of Windows API system function call frequencies observed through the behavioral analysis of malicious and benign software provides machine learning models which have higher real-world accuracy due to an understanding of the differences between malicious and benign behaviors.
- We provide an analysis of models created by several linear and non-linear machine learning algorithms along with different datasets of malicious and benign software samples to examine their relative efficacy in the classification of malicious and benign software through the evaluation of Windows API call frequencies.

Section II presents an overview of related work. Section III describes the malware behavior analysis methodology we employed including the hardware and software setup, malware dataset, and analysis of the observed API calls. Section IV discusses our machine learning approach, including an explanation of the algorithms used and a comparison of the accuracy

This research is supported by the National Science Foundation(NSF), USA, Award #1739032.

of the resulting models. Finally, Section V concludes the paper.

II. Related Work

Machine learning is widely used in the field of cybersecurity and there are a number of different machine learning algorithms available for research [8], including decision tree [13] and logistic regression [7], to name but a few. Static analysis of malware involves inspection of the code at rest and has been shown to be successful in the classification of malware family [15]. This includes the examination of the register, operation codes, Portable Executable structured information and more. Dynamic analysis has proven effective in cases where static analysis would fail due to encryption or dynamic code loading [11], making this approach more attractive for the extraction of features for machine learning.

Dynamic analysis typically involves the use of a sandbox environment, such as Cuckoo Sandbox [1] which reports behaviors in terms of system API calls. Smith et al. [14] demonstrated the potential for understanding malicious API calls through machine learning algorithms. The large number of Windows API calls found in malicious as well as benign software samples frustrates the process of feature selection for machine learning algorithms. One approach is to categorize the function calls based on their general function [12] [9] and then evaluate the entropy of these categorial functions based on Information Gain [3], which essentially is a measure of how much information a randomly chosen data element in a set will teach us about another randomly chosen element in a set. Heuristic N-Grams analysis also adopts the Information Gain technique and has been shown to be effective in distinguishing malware from benign software [6]. This shows that while both benign and malicious software perform many of the same Windows API calls, their relative frequencies are distinguishable.

Another approach for selecting which Windows API calls to use as features involves narrowing the scope of analyzed malware samples to model specific malware families, such as WannaCry ransomware [5]. Malware family classification can be enhanced with machine learning models, as shown in [9], however here we also see the same issues with feature extraction and definition. Malware authors are aware of the attempts of researchers and system defenders to identify malicious software and often employ anti-analysis features [10]. As a result there has been research into image processing with Deep Learning – in this way, machine learning has been used in malware classification based upon image processing using an extracted local binary pattern [4]. There is no currently defined methodology for accurate, non-biased feature extraction of malicious behavior observed through dynamic analysis; therefore it is not reasonable to assume that bias is restricted without a systematic approach for the comparison of machine learning models with differing datasets.

III. Malware Behavior Analysis

In order to programmatically observe the behavior of malware in an isolated environment, we designed an environment to allow for the installation of Cuckoo and the analysis of known malware samples in a virtual machine sandbox per the installation instructions provided by Cuckoo [1]. Our goal was to focus on the evaluation of potentially malicious software affecting Windows operating systems.

A. Setup and Malware Dataset

Cuckoo was configured per the installation guide found on the Cuckoo website [1], including two 64-bit Windows 7 virtual machines installed on the Ubuntu host. Cuckoo supports many virtualization software solutions but does assume the usage of VirtualBox by default, so for ease of setup we chose this platform. VirtualBox is a free system virtualization product developed by Oracle and it easily integrates with Cuckoo for administration of the virtual machines.

Known malware samples were acquired from Malpedia [2], a curated online resource of malicious software containing multiple versions of malware samples seen over time. This allows for the observation of evolving behaviors as the methods of exploiting system and application vulnerabilities changes with new generations of malware. Malpedia samples often include references to third party analysis of the malware as well as identified malware family and threat actor affiliation. This information is quite valuable for those desiring to create custom signatures in Cuckoo for malware family attribution.

B. Methodology

Once the analysis has been performed, Cuckoo generates a report of the observed activity, including but not limited to changes to the registry, newly spawned processes, file creation and access, virtual memory access, HTTP communication to an external IP, and much more. These behavioral events are captured as a number of Windows API calls and can be referenced programmatically through created JSON files.

One example of a Windows API call is the GetComputerNameW function. The activity shown here is the usage of the GetComputerNameW Windows function which retrieves the NetBIOS name of the local computer. The signature matched is identified as “antivm_queries_computername” and the function of this call is indeed to query for the name of the computer. For the full set of malware samples we analyzed, we found that this particular API call was committed a total of 38,867 times.

IV. Machine Learning Approach

Our dataset consists of the Windows application programming interface (API) function calls observed by our Cuckoo malware behavioral analysis environment as described in Section III. The API calls recorded in our dataset represent the activities performed by 7,401 malware samples. Windows API functions are called by applications in order to operate in a Windows environment [20]. Analysis of API calls made by an application in a Windows environment therefore presents a concrete record of all behaviors performed by that application.

In our behavioral analysis we identified a number of Windows API calls and their frequencies which correspond

to the actions performed by malware on a system, such as registry changes, code injection into running processes, file modification, etc. It is important to note that it is not trivial to analyze API calls for malicious behavior. In addition to known malware samples, we also performed behavioral analysis against a set of thirty known benign software samples.

These benign software samples consist of a mixture of application installers, Java applications, Microsoft Word documents, and similar executable files. Analysis of malicious and benign software samples in Cuckoo resulted in the observation of over 138 million API calls, with 264 unique referenced functions. Of these 264 API calls we found that 84 unique API functions were found in malware but not in benign software.

Unfortunately, these 84 APIs were called in only a small fraction of the malware samples and therefore are not useful as a "smoking gun" for determining if a given application is malicious if its behavior includes these particular function calls. Therefore, we concluded that the best usage for applying a machine learning model in malware analysis would be to examine the relative frequencies of these 264 APIs across malicious and benign software. It is important to note that the frequency of the API calls largely vary. For example, we found a greater number of the "GetAsyncKeyState" Windows API call in comparison to the "GetCursorPos" call. This is likely due to a greater interest in software to react to a computer user's mouse button usage [18] than in the screen coordinates of the mouse cursor [19]. This suggests that while particular Windows API functions may be selected as features in a machine learning model to identify malware, there is reason to analyze what such a decision really means for understanding the behavior of the identified malicious software. Our approach was to apply several machine learning algorithms to three different sized malware sample sets and compare them. Our goal was to identify the algorithmic approach which should most reliably classify malicious software from benign software through analysis of the frequency of Windows API calls.

A. Methodology

Eight machine learning algorithms were evaluated on the set of values describing the Windows APIs called for each malicious and benign sample:

- Logistic Regression (LR)
- Linear Discriminant Analysis (LDA)
- K-Nearest Neighbors (KNN)
- Classification and Regression Trees (CART)
- Gaussian Naive Bayes (NB)
- Support Vector Machines (SVM)
- Decision Tree
- Random Forest

Our desire was to use a mixture of linear (LR and LDA) and non-linear (KNN, CART, NB, and SVM) algorithms to determine which would be good for our dataset. The applicability of classification trees to relationship models led us to further break CART down into Decision Tree and Random Forest classifiers using the Scikit-Learn Python library for visualization as shown in [17]. Our implementation included Python and the SciPy platform using a random number seed which was reset before each run to ensure that the results were directly comparable as shown in [16].

Through the use of a Python program we were able to compare each of these algorithms as they attempted to classify a given software sample as malicious or benign through the evaluation of the relationships between the API frequencies in known malicious and benign software samples. Once such a training model was created for each algorithm, these models were used to classify a testing subset of unevaluated software. The accuracy of the training models against the test subset for each algorithm was then compared to determine the best performing algorithm. In addition to evaluating different machine learning algorithms, the decision was made to evaluate each upon different sample sizes of malicious and benign software. The Large Sample dataset contains the API frequencies of all 7,400 malware samples from our known malware dataset along with the API frequencies of thirty known benign software samples. The Medium Sample dataset consists of 853 randomly selected known malware samples along with our thirty known benign software samples. The Small Sample dataset consists of an equal number of known malicious and known benign samples, again with a randomly selected subset of the original 7,400 malware samples. Table I summarizes the three datasets used to test the accuracy of the algorithms against different ratios of known malicious to known benign software.

TABLE I
THREE DATASETS

Dataset	Malware Count	Benign Count
Large Sample	7,400	30
Medium Sample	853	30
Small Sample	30	30

We utilized a 70/30 split for training and validation for each dataset, which trains each machine learning model on 70% of the dataset and then tests 30% to determine accuracy. 10-fold cross validation was used to estimate the accuracy of the machine learning algorithms.

Given its high relative accuracy across our datasets, we chose to further evaluate our models generated with KNN to break down their confusion matrices. A confusion matrix considers the total number of elements in the validation dataset and then further classifies these elements into the following categories: true positive, false positive, true negative, and false positive. Since we are attempting to classify malware, a true positive result would reflect that an actual malware was identified as malicious. Likewise, a false positive would involve the classification of benign software as malicious. Therefore, our most successful models would generate the highest number of true positives and true negatives.

Additionally, we also observed high accuracy with CART. This intrigued us due to the possibility of relationship modeling using decision trees and lead us to include a deeper dive into the accuracy of different decision tree models. Our Decision Tree implementation utilizes two methods for determining a root node with the goal of comparing these

two methods for different accuracies. Equation 1 describes the Gini Index, which is a metric used to determine how often a randomly chosen feature would be incorrectly identified. In our case, each API frequency is evaluated as a feature and the one with the lowest Gini Index is identified as an appropriate root node for a tree.

$$Gini(E) = 1 - \sum_{j}^{c} P_j^2 \tag{1}$$

Equation 2 describes our second method for determining a root node for a tree. Here entropy is used to determine the impurity of a feature as a measure of information gain. In this way the feature with highest calculated entropy will be selected as the root node of a tree.

$$H(X) = -\sum_{j}^{c} p(x_i) log_2 p(x_i) \tag{2}$$

B. Results

1) Large Sample: Table II shows most of our algorithms are extremely close to purporting 100% accuracy. This is likely due to the much larger number of malware samples included in this dataset skewing the learning algorithm's perception to understand mostly malicious behavior. This leads us to conclude that four models for the Large Sample are overfit, with the notable exception of NB which appears to underperform for each of our datasets. We also observed that KNN gave us 9 false positives and 2,220 true positives according to the model. Figure 1 illustrates the Decision Tree model

TABLE II
COMPARISON OF ALGORITHMS FOR LARGE SAMPLE

Linear/Non-linear	Algorithm	Mean	Standard Deviation
Linear	LR	0.995193	0.002470
Linear	LDA	0.981541	0.010784
Non-linear	KNN	0.995963	0.002499
Non-linear	CART	0.995001	0.002878
Non-linear	NB	0.241691	0.023089
Non-linear	SVM	0.995963	0.002499

for the Large Sample as a series of branching nodes. Each node describes a value for a particular Windows API, followed by the Gini Index value, the number of software samples (malicious or benign) which made a call to this particular Windows API, a tuple which delineates the number of benign software from the number of malicious software samples, and finally a classification of Malware or Benign as predicted by the learning model.

The Decision Tree model for the Large Sample reported 99.59% accuracy with gini index with one true negative, three false negatives, eight false positives, and 2,217 true positives. We also observed zero true negatives, zero false negatives, nine false positives, and 2,220 true positives with 99.6% accuracy using entropy. Utilizing Random Forest for the Large Sample we observed two true negatives, zero false negatives, seven false positives, and 2,220 true positives with 99.69% accuracy.

The decision trees created show a pronounced bias toward identifying malware as opposed to identifying benign software samples. This is likely due to the much higher amount of malware samples in the Large Sample. This apparent overfit is what led us to continue evaluating these algorithms with a smaller dataset.

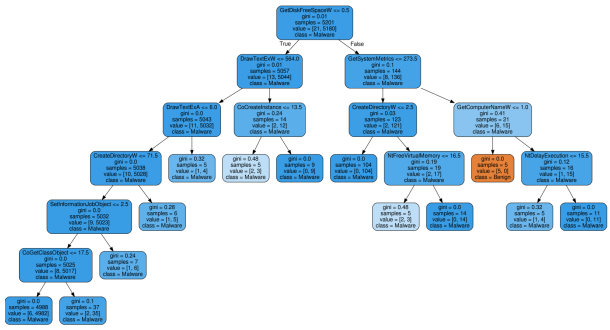

Fig. 1. Decision Tree for Large Sample

2) Medium Sample: In the Medium Sample we observe results which appear to be much less overfit than what was seen from the Large sample. Table III shows a much different algorithm comparison than what was seen for the Large Sample. Of interest is that NB performed similarly to the Large Sample, with a much lower accuracy than the other algorithms. As with the Large Sample we see that KNN and SVM perform with the highest accuracy. The confusion matrix for KNN shows us that this algorithm gave us one true negative, three false negatives, ten false positives and 251 true positives according to the model. Our Decision Tree model

TABLE III
COMPARISON OF ALGORITHMS FOR MEDIUM SAMPLE

Linear/Non-linear	Algorithm	Mean	Standard Deviation
Linear	LR	0.954654	0.027075
Linear	LDA	0.881914	0.058108
Non-linear	KNN	0.964410	0.020299
Non-linear	CART	0.959572	0.025247
Non-linear	NB	0.737916	0.060903
Non-linear	SVM	0.966023	0.019854

decreased accuracy by approximately 8% using Gini, with four true negatives, sixteen false negatives, seven false positives, and 238 true positives. This is in contrast to our model using entropy, which decreased by approximately 2% with nine true negatives, four false negatives, two false positives, and 250 true positives. This suggests that the use of entropy to determine a root node works much better than the Gini approach for the API frequency data being categorized. Our Random Forest model decreased accuracy by approximately 4% with one true negative, one false negative, ten false positives, and 253 true positives. The lack of true negatives in this result suggest that either a small amount of benign software was provisioned into the test dataset or this model continues to suffer from overfit.

Figure 3 illustrates a decision tree generated by our model for the Medium Sample. Here we see quite a difference from

the tree for the Large Sample, as there is much less bias toward identifying malware versus benign software. This tree appears to not suffer from as much overfit as we saw in the Large Sample. As a whole, the machine learning algorithms created models which seem to be more accurate for the Medium Sample than for the Large Sample.

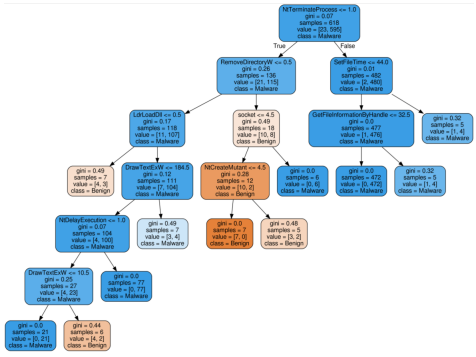

Fig. 2. Decision Tree for Medium Sample

3) Small Sample: For the Small Sample we observe that our models are underfit. We have a proportionately high ratio of true negatives to true positives reported, along with a relatively higher number of false negatives. The overall accuracy of our models has reduced significantly as a result. Table IV shows a greater amount of outliers as opposed to our previous sample datasets, reducing the overall accuracy. Of interest is that NB has increased in accuracy in comparison to its performance in the larger datasets while others, especially SVM have greatly reduced accuracy. As noted earlier, KNN and SVM had previously performed with comparable accuracy. Here we can see that KNN performed with 36% greater accuracy than SVM. Furthermore, KNN gave us eight true negatives, six false negatives, zero false positives and four true positives according to the model. Interestingly, as seen

TABLE IV
COMPARISON OF ALGORITHMS FOR SMALL SAMPLE

Linear/Non-linear	Algorithm	Mean	Standard Deviation
Linear	LR	0.695	0.180901
Linear	LDA	0.67	208806
Non-linear	KNN	0.715	0.264622
Non-linear	CART	0.81	0.185472
Non-linear	NB	0.79	0.115758
Non-linear	SVM	0.355	0.180901

with the Smaller Sample we note that here our decision trees created with entropy have greater accuracy than those using the Gini Index. Our decision tree made with Gini had an accuracy of approximately 83.33% with a confusion matrix showing seven true negatives, two false negatives, one false positive, and eight true positives. Comparatively, our model using entropy resulted in an accuracy of approximately 94.44% along with eight true negatives, one false negative, zero false positives, and nine true positives. This suggests that as the sample datasets become smaller and the ratio between malware and benign software samples becomes more equal, entropy is a much better deciding measure for a root node. Furthermore, our Random Forest model suffers from reduced accuracy with this Small Sample dataset. Eight true negatives and eight true positives are offset by two false negatives and zero false positives. There is a progressive increase in false negatives as our sample size and disparity between malicious and non-malicious software samples has decreased. Our Decision Tree for the Small Sample illustrates a bias toward classifying software as benign. This indicates the increase in false negatives as our models are now clearly underfit.

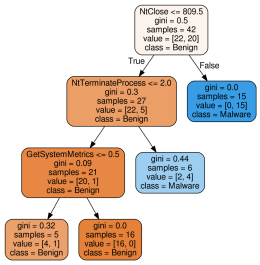

Fig. 3. Decision Tree for Small Sample

C. Discussion

Our research shows that the analysis of the varying frequencies of Windows API calls made between malicious and benign software can be used for classification with extremely high accuracy depending on the machine algorithm used and characteristics of the dataset. Our Large Sample dataset (reported 99% accuracy) trained the models to recognize the most malicious activity from the features because the data was highly biased toward understanding the Windows API call frequencies as malicious behavior. Our Small Sample dataset (reported 88% accuracy) contained an equal number of malicious and benign software samples, resulting in a bias toward understanding the feature properties as benign indicators – likely due to a higher number of benign samples in the training dataset for the model not producing enough distinguishing features between the classes. Using the Goldilocks analogy, the Medium Sample dataset (reported 96% accuracy) was a "just right" middle-ground which captured more true negatives and true positives for malware classification based solely on Windows API call frequency.

Our research also shows that the malware analysis machine learning models generated by various algorithms are subject to varying accuracy depending on the data used to build those models as well as the algorithms selected. For example, Naive-Bayes did not produce strong models for us, possibly due to the conditional independence assumption inherent in the algorithm. Furthermore, decision trees trained on our dataset using entropy to determine the root note were much more accurate than those using Gini. Indeed, our research suggests that there is no "one size fits all" for malware detection using machine learning. As noted previously, recent work has shown the efficacy of using machine learning algorithms to identify malware based upon observed behavior through

dynamic analysis. Our research supports the viability of this approach for malware detection, with the added caveat that it is simple to overfit or underfit machine learning models to the malware behavior data. Considering that there is currently no universal dataset for malware behavior or a recognized unbiased approach to creating models in a consistent manner, we find that it is imperative for researchers to employ exhaustive investigation into varied machine learning algorithms and techniques when evaluating the efficacy of a model's ability to identify malware so as to not fall victim to the bias of a given dataset or methodology.

We also believe it is worthwhile to consider not simply the accuracy score of a machine learning algorithm but also what meaning can be derived from the produced model. Decision trees in particular give us information about the relation between features when observed visually. The differences in the trees described in Figures 9, 14, and 19 provide an opportunity to analyze the logic employed by the machine learning algorithm in developing the model. As the model learns to understand malicious behavior, we have discovered the possibility of using the relationship between feature values as an indication of how to understand what patterns exist in the dynamic analysis of malware. This suggests that there is potential for fingerprinting dynamically observed malicious activity based upon common Windows API calls made which perform actions that are typically malicious in nature because they serve a typically malicious purpose. This is a different approach from recognizing the frequencies of API calls made by malware in that we would not be concerned with just one or more function calls - our concern would be in what these function calls tell us about the malicious or benign intent of the software.

V. Conclusion

We have analyzed two linear and four non-linear machine learning algorithms and compared their relative accuracies against datasets with different proportions of malicious and benign software. In this way, we have shown that the efficacy of a machine learning model is dependent upon the machine learning algorithm and the type of data that model is built upon. Different machine learning models provide differing results and it would appear that some are better than others for analyzing system calls as features. Malicious and benign software share many of the same behaviors – read/write data, open/close files, attempt network connections, etc., albeit in different relative frequencies. We have shown that these frequencies can be modeled to give us the ability to distinguish malware from benign software as well as help us to understand the relationship between these behaviors through analysis of decision trees.

References

[1] Cuckoo Foundation. "Automated Malware Analysis." Cuckoo Sandbox - Automated Malware Analysis. 2014. Accessed March 10, 2019. https://cuckoosandbox.org/.

[2] Plohmann, Daniel, M. Clauß, S. Enders, and E. Padilla. "Malpedia: a collaborative effort to inventorize the malware landscape." Proceedings of the Botconf (2017).

[3] Gandotra, Ekta, Divya Bansal, and Sanjeev Sofat. "Zero-day malware detection." In 2016 Sixth International Symposium on Embedded Computing and System Design (ISED), pp. 171-175. IEEE, 2016.

[4] Luo, Jhu-Sin, and Dan Chia-Tien Lo. "Binary malware image classification using machine learning with local binary pattern." In 2017 IEEE International Conference on Big Data (Big Data), pp. 4664-4667. IEEE, 2017.

[5] Chen, Qian, and Robert A. Bridges. "Automated behavioral analysis of malware: A case study of wannacry ransomware." In 2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA), pp. 454-460. IEEE, 2017.

[6] Darshan, SL Shiva, MA Ajay Kumara, and C. D. Jaidhar. "Windows malware detection based on cuckoo sandbox generated report using machine learning algorithm." In 2016 11th International Conference on Industrial and Information Systems (ICIIS), pp. 534-539. IEEE, 2016.

[7] Kumar, B. Jyothi, H. Naveen, B. Praveen Kumar, Sai Shyam Sharma, and Jaime Villegas. "Logistic regression for polymorphic malware detection using ANOVA F-test." In 2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS), pp. 1-5. IEEE, 2017.

[8] Liu, Qiang, Pan Li, Wentao Zhao, Wei Cai, Shui Yu, and Victor CM Leung. "A survey on security threats and defensive techniques of machine learning: A data driven view." IEEE access 6 (2018): 12103-12117.

[9] Pektaş, Abdurrahman, and Tankut Acarman. "Malware classification based on API calls and behaviour analysis." IET Information Security 12, no. 2 (2017): 107-117.

[10] Jain, Aruna, and Akash Kumar Singh. "Integrated Malware analysis using machine learning." In 2017 2nd International Conference on Telecommunication and Networks (TEL-NET), pp. 1-8. IEEE, 2017.

[11] Feng, Pengbin, Jianfeng Ma, Cong Sun, Xinpeng Xu, and Yuwan Ma. "A Novel Dynamic Android Malware Detection System With Ensemble Learning." IEEE Access 6 (2018): 30996-31011.

[12] Daku, Hajredin, Pavol Zavarsky, and Yasir Malik. "Behavioral-Based Classification and Identification of Ransomware Variants Using Machine Learning." In 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), pp. 1560-1564. IEEE, 2018.

[13] Roseline, S. Abijah, and S. Geetha. "Intelligent Malware Detection using Oblique Random Forest Paradigm." In 2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI), pp. 330-336. IEEE, 2018.

[14] Smith, Michael, Joey Ingram, Christopher Lamb, Timothy Draelos, Justin Doak, James Aimone, and Conrad James. "Dynamic Analysis of Executables to Detect and Characterize Malware." In 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), pp. 16-22. IEEE, 2018.

[15] Sun, Bowen, Qi Li, Yanhui Guo, Qiaokun Wen, Xiaoxi Lin, and Wenhan Liu. "Malware family classification method based on static feature extraction." In 2017 3rd IEEE International Conference on Computer and Communications (ICCC), pp. 507-513. IEEE, 2017.

[16] Brownlee, Jason. "Your First Machine Learning Project in Python Step-By-Step." Machine Learning Mastery. March 04, 2019. Accessed March 10, 2019. https://machinelearningmastery.com/machine-learning-in-python-step-by-step/.

[17] Koehrsen, Will. "How to Visualize a Decision Tree from a Random Forest in Python Using Scikit-Learn." Towards Data Science. August 19, 2018. Accessed March 10, 2019. https://towardsdatascience.com/how-to-visualize-a-decision-tree-from-a-random-forest-in-python-using-scikit-learn-38ad2d75f21c.

[18] Windows-Sdk-Content. "GetAsyncKeyState Function." Microsoft Docs. December 04, 2018. Accessed March 10, 2019. https://docs.microsoft.com/en-us/windows/desktop/api/winuser/nf-winuser-getasynckeystate.

[19] Windows-Sdk-Content. "GetCursorPos Function." Microsoft Docs. December 04, 2018. Accessed March 10, 2019. https://docs.microsoft.com/en-us/windows/desktop/api/winuser/nf-winuser-getcursorpos.

[20] Kennedy, John, Michael Satran, and Mark LeBlanc. "API Index - Windows Applications." Windows Applications — Microsoft Docs. May 30, 2018. Accessed March 23, 2019. https://docs.microsoft.com/en-us/windows/desktop/apiindex/api-index-portal.